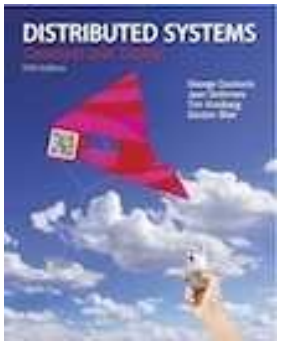
# Slides for Chapter 2: System Models




From **Coulouris, Dollimore, Kindberg and Blair**

**Distributed Systems: Concepts and Design**

Edition 5, © Addison-Wesley 2012

Dr. Ibrahim Al-Baltah

# Introduction

- Each type of model is intended to provide an abstract, simplified but consistent description of a relevant aspect of distributed system design:
  - ✓Physical models
  - ✓Architectural models
  - ✓Fundamental models

# Physical models

➢A physical model is a representation of the underlying hardware elements of a distributed system that abstracts away from specific details of the computer and networking technologies employed.

# Figure 2.1
# Generations of distributed systems

| *Distributed systems:* | *Early* | *Internet-scale* | *Contemporary* |
|---|---|---|---|
| *Scale* | Small | Large | Ultra-large |
| *Heterogeneity* | Limited (typically relatively homogenous configurations) | Significant in terms of platforms, languages and middleware | Added dimensions introduced including radically different styles of architecture |
| *Openness* | Not a priority | Significant priority with range of standards introduced | Major research challenge with existing standards not yet able to embrace complex systems |
| *Quality of service* | In its infancy | Significant priority with range of services introduced | Major research challenge with existing services not yet able to embrace complex systems |

# Architectural models

- The architecture of a system is its structure in terms of separately specified components and their interrelationships.
- The overall goal is to ensure that the structure will meet present and likely future demands on it.
- Major concerns are to make the system reliable, manageable, adaptable and cost-effective.

# Architectural elements

- To understand the fundamental building blocks of a distributed system, it is necessary to consider four key questions:
  - What are the entities that are communicating in the distributed system?
  - How do they communicate, or, more specifically, what communication paradigm is used?
  - What (potentially changing) roles and responsibilities do they have in the overall architecture?
  - How are they mapped on to the physical distributed infrastructure (what is their placement)?

# Communication paradigms

- three types of communication paradigm:
  1) interprocess communication
  2) remote invocation
  3) indirect communication

# Communication paradigms

- Remote invocation: represents the most common communication paradigm in distributed systems
- Request-reply protocols: Request-reply protocols are effectively a pattern imposed on an underlying message-passing service to support client-server computing.
- Remote procedure calls: In RPC, procedures in processes on remote computers can be called as if they are procedures in the local address space.
- Remote method invocation: With this approach, a calling object can invoke a method in a remote object.

# Communication paradigms

- Group communication: Group communication is concerned with the delivery of messages to a set of recipients and hence is a multiparty communication paradigm supporting one-to-many communication.
- Publish-subscribe systems: is used when a large number of producers (or publishers) distribute information items of interest (events) to a similarly large number of consumers (or subscribers).
- Message queues: Whereas publish-subscribe systems offer a one-to-many style of communication, message queues offer a point-to-point service whereby producer processes can send messages to a specified queue and consumer processes can receive messages from the queue or be notified of the arrival of new messages in the queue.

# Communication paradigms

- Distributed shared memory: Distributed shared memory (DSM) systems provide an abstraction for sharing data between processes that do not share physical memory.

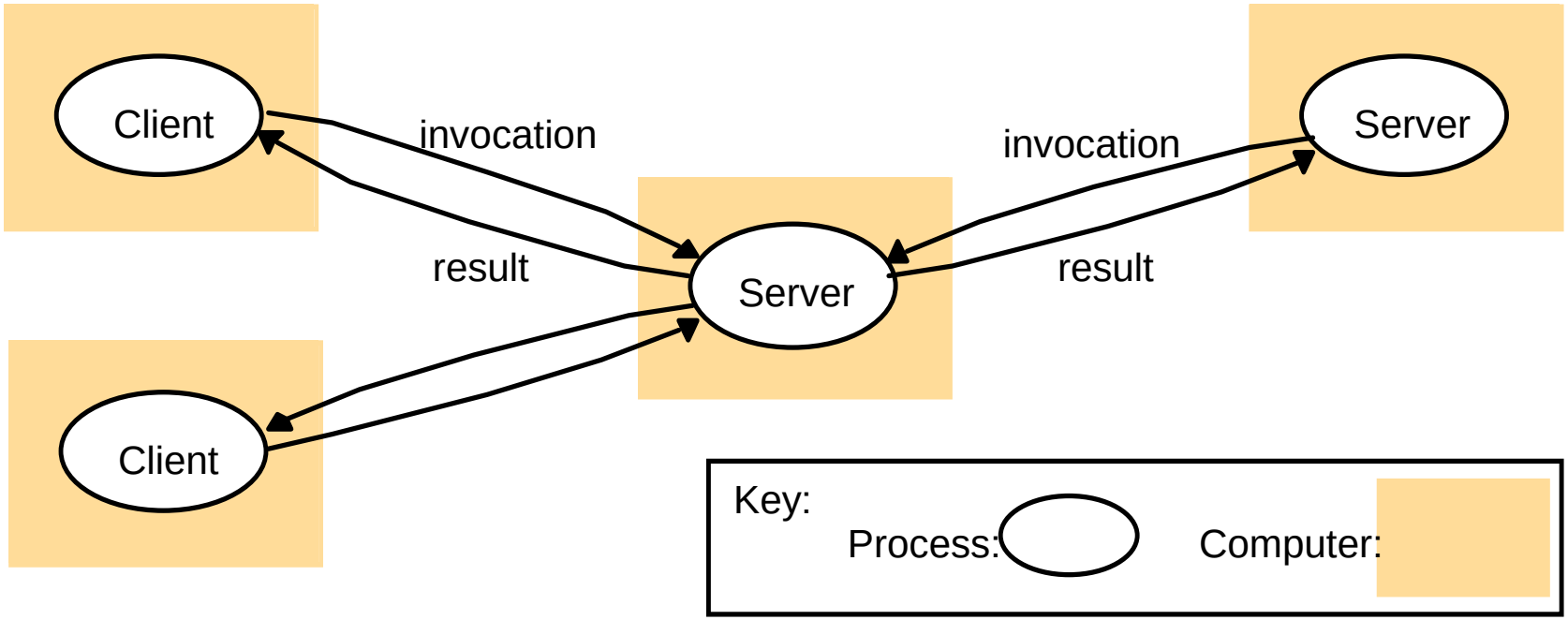
# Figure 2.2
# Communicating entities and communication paradigms

| *Communicating entities (what is communicating)* | | *Communication paradigms (how they communicate)* | | |
|---|---|---|---|---|
| *System-oriented entities* | *Problem-oriented entities* | *Interprocess communication* | *Remote invocation* | *Indirect communication* |
| Nodes | Objects | Message passing | Request-reply | Group communication |
| Processes | Components | Sockets | RPC | Publish-subscribe |
| | Web services | Multicast | RMI | Message queues |
| | | | | Tuple spaces |
| | | | | DSM |

# Client-server

- This is the architecture that is most often cited when distributed systems are discussed.
- Servers may in turn be clients of other servers, as the figure indicates. For example, a web server is often a client of a local file server that manages the files in which the web pages are stored.

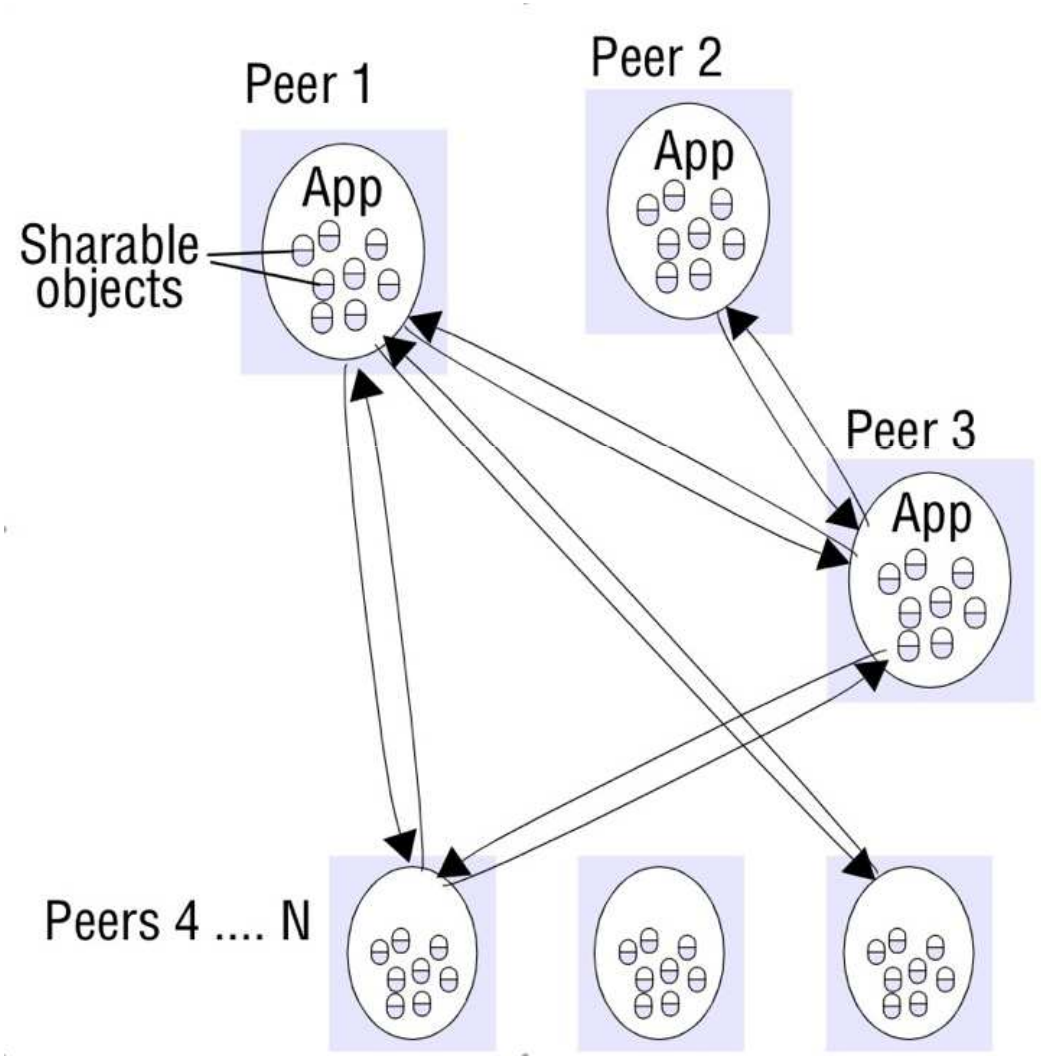
# Figure 2.3
# Clients invoke individual servers



# Peer-to-peer

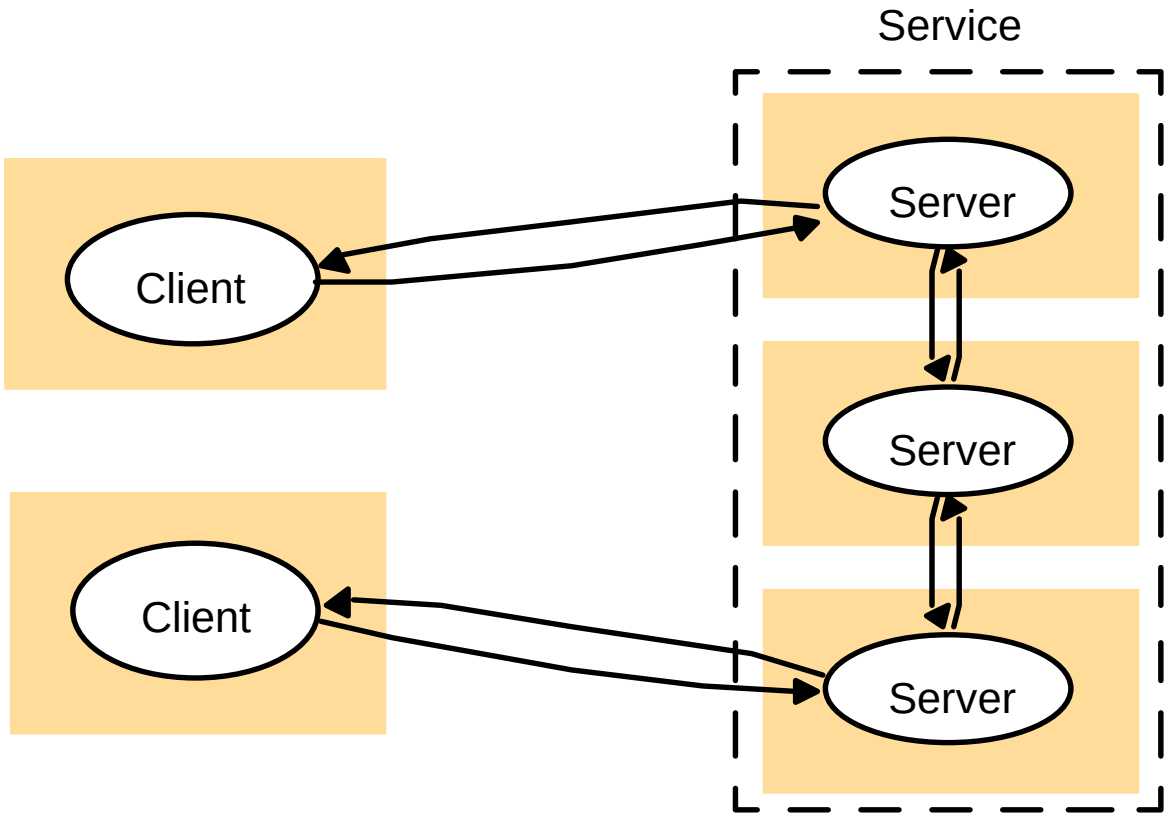
- In this architecture all of the processes involved in a task or activity play similar roles.
- There is no any distinction between client and server

# Figure 2.4a
# Peer-to-peer architecture



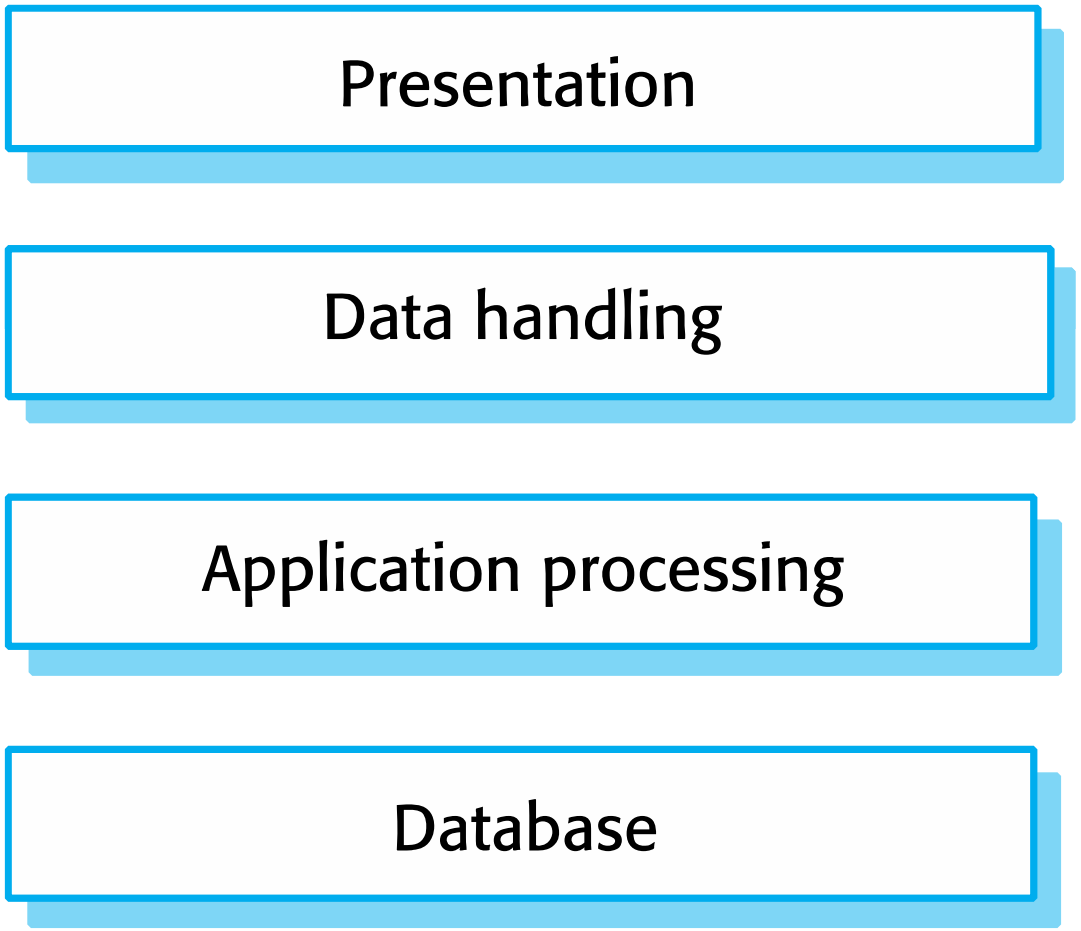
# Mapping of services to multiple servers

- Services may be implemented as several server processes in separate host computers interacting as necessary to provide a service to client processes.




A service provided by multiple servers

# Layered architectural model for client–server applications



# Layers in a client/server system

- ***Presentation***

  concerned with presenting information to the user and managing all user interaction.

- ***Data handling***

  manages the data that is passed to and from the client. Implement checks on the data, generate web pages, etc.

- Application processing

  concerned with implementing the logic of the application and so providing the required functionality to end users.

- Database

  Stores data and provides transaction management services, etc.

# Architectural patterns

Widely used ways of organizing the architecture of a distributed system:

***Master-slave architecture***, which is used in real-time systems in which guaranteed interaction response times are required.

***Two-tier client-server architecture***, which is used for simple client-server systems, and where the system is centralized for security reasons.

***Multi-tier client-server architecture***, which is used when there is a high volume of transactions to be processed by the server.

***Distributed component architecture***, which is used when resources from different systems and databases need to be combined, or as an implementation model for multi-tier client-server systems.

***Peer-to-peer architecture***, which is used when clients exchange locally stored information and the role of the server is to introduce clients to each other

# Master-slave architectures

- Master-slave architectures are commonly used in real-time systems where there may be separate processors associated with data acquisition from the system's environment, data processing and computation and actuator management.
- The 'master' process is usually responsible for computation, coordination and communications and it controls the 'slave' processes.
- 'Slave' processes are dedicated to specific actions, such as the acquisition of data from an array of sensors.

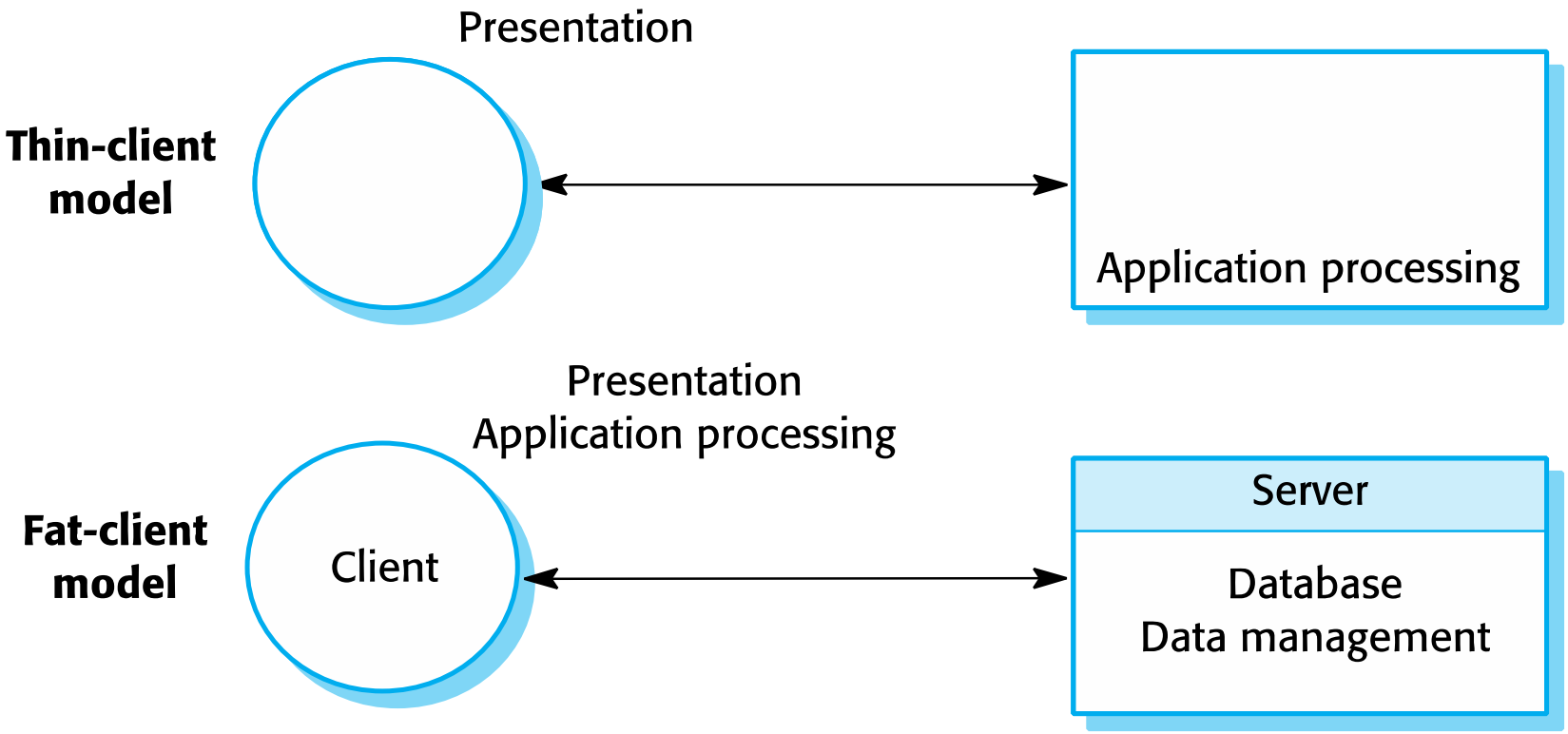
# Two-tier client server architectures

- In a two-tier client-server architecture, the system is implemented as a single logical server plus an indefinite number of clients that use that server.

Thin-client model, where the presentation layer is implemented on the client and all other layers (data management, application processing and database) are implemented on a server.

Fat-client model, where some or all of the application processing is carried out on the client. Data management and database functions are implemented on the server.

# Thin- and fat-client architectural models



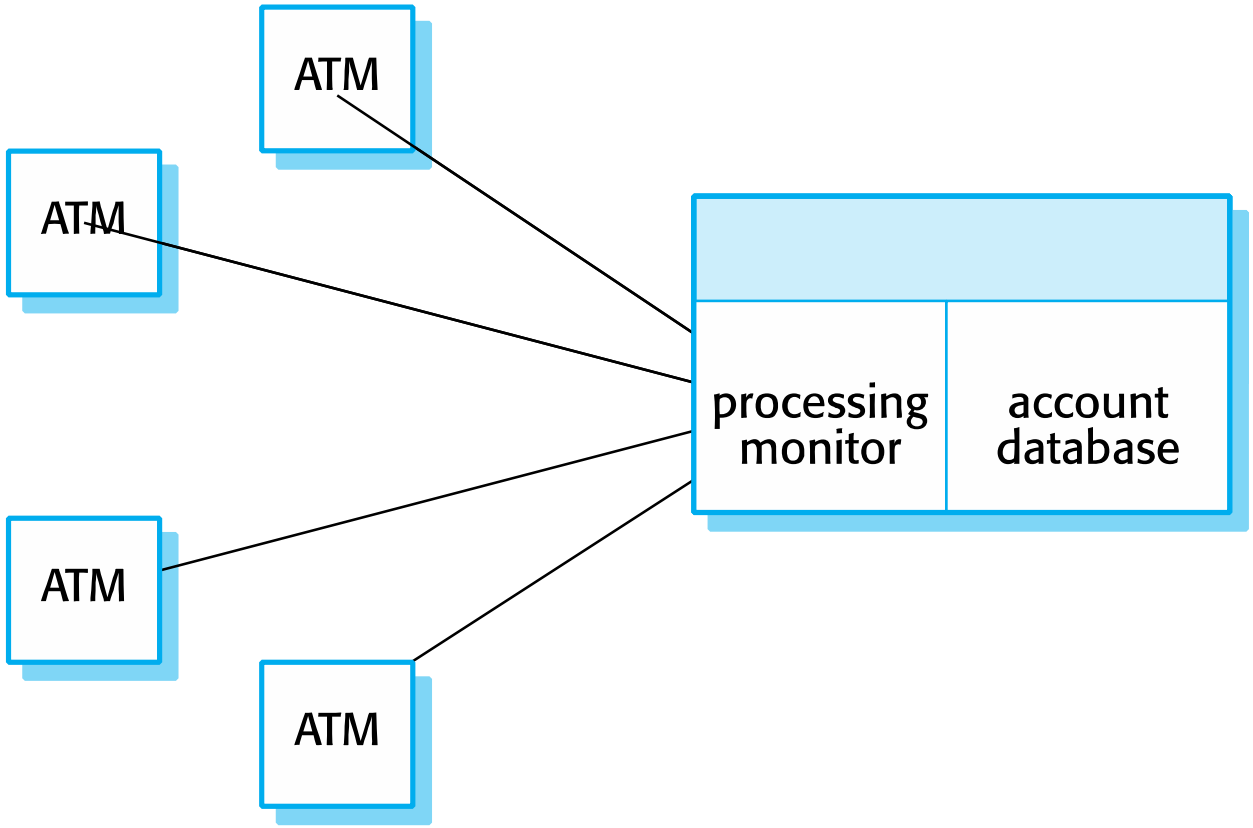
# Thin client model

- Used when legacy systems are migrated to client server architectures.
  - The legacy system acts as a server in its own right with a graphical interface implemented on a client.
- A major disadvantage is that it places a heavy processing load on both the server and the network.

# Fat client model

- More processing is delegated to the client as the application processing is locally executed.
- Most suitable for systems where the capabilities of the client system are known in advance.
- More complex than a thin client model especially for management. New versions of the application have to be installed on all clients.
- Example: ATM

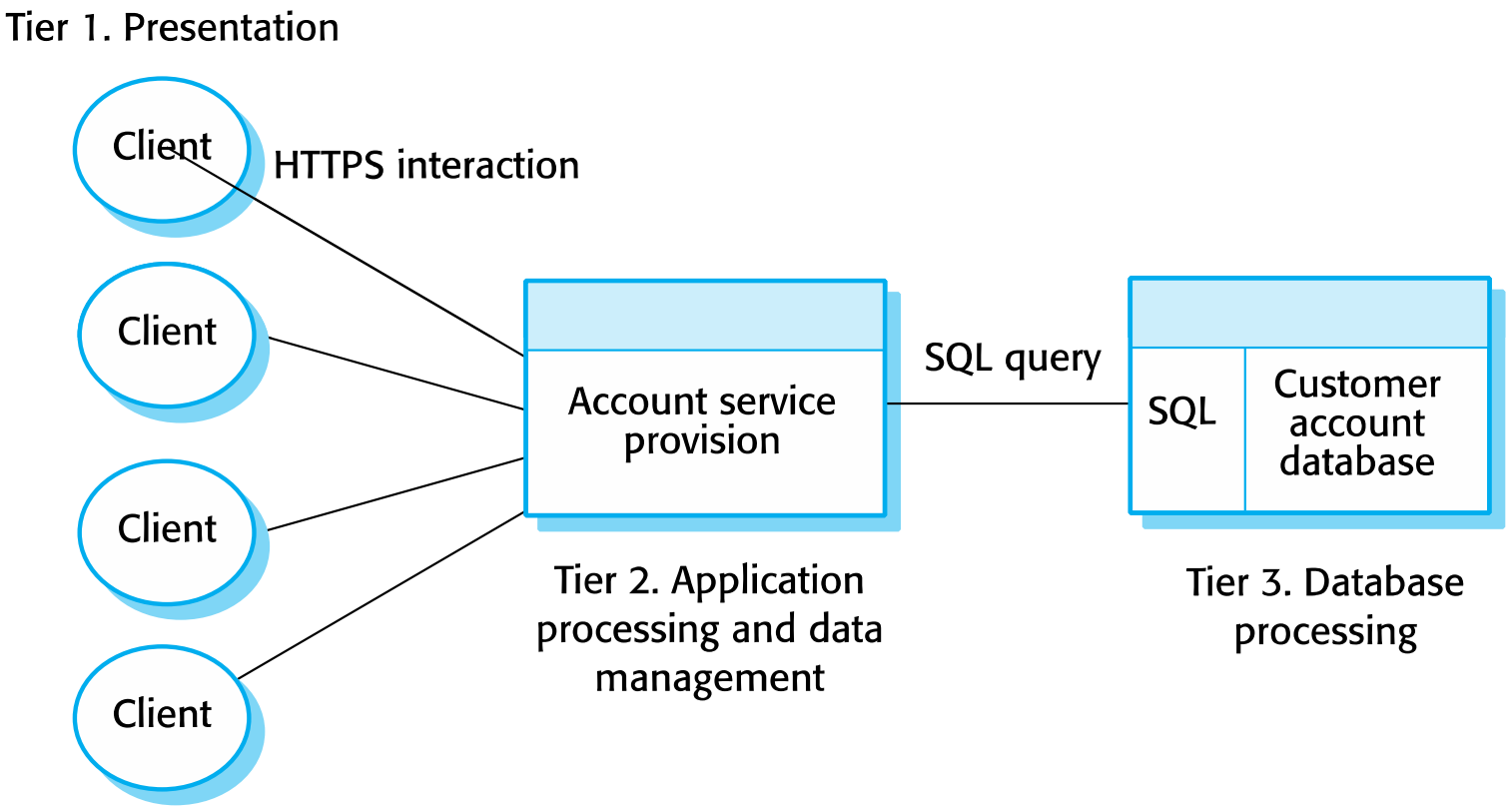
# A fat-client architecture for an ATM system



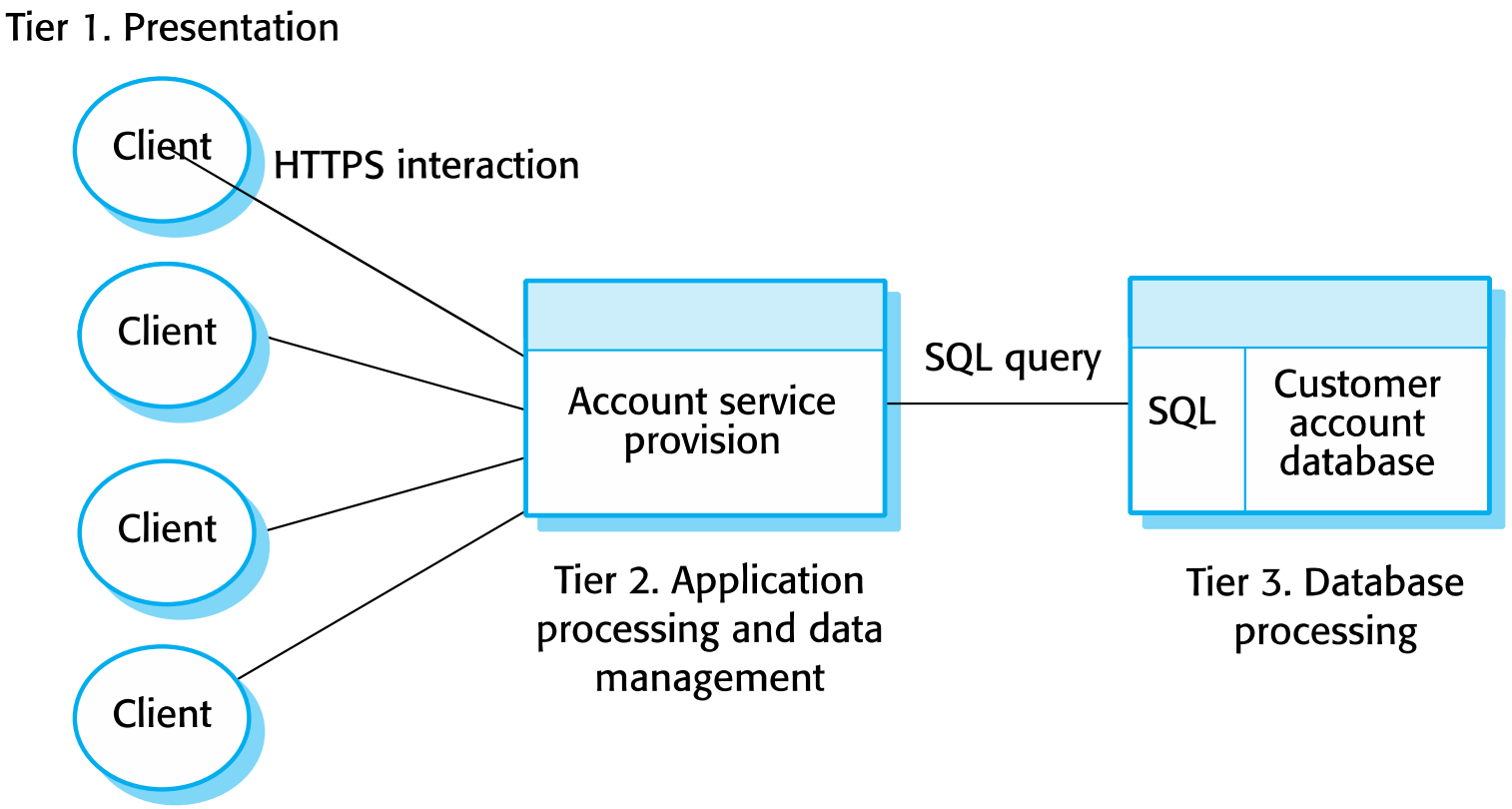
# Multi-tier client-server architectures

- In a ‘multi-tier client–server’ architecture, the different layers of the system, namely presentation, data management, application processing, and database, are separate processes that may execute on different processors.
- This avoids problems with scalability and performance if a thin-client two-tier model is chosen, or problems of system management if a fat-client model is used.
- Example: Internet banking system

# Three-tier architecture for an Internet banking system



# Three-tier architecture for an Internet banking system



# References

1- Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design 5th edition, Pearson Education 2012

2- Ian Sommerville, 2014. Software Engineering, 10th edition, Addison-Wesley